2014 ASHRAE/IBPSA-USA
Building Simulation Conference
Atlanta, GA
September 10-12, 2014

LINKING SIMULATION PROGRAMS, ADVANCED CONTROL AND FDD ALGORITHMS WITH A BUILDING MANAGEMENT SYSTEM BASED ON THE FUNCTIONAL MOCK-UP INTERFACE AND THE BUILDING AUTOMATION JAVA ARCHITECTURE STANDARDS

Thierry Stephane Nouidui, Michael Wetter
Simulation Research Group, Building Technology and Urban Systems Department
Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA

ABSTRACT

This paper describes the extension of a building management system with an interface that allows the import of simulation models, control, and fault detection (FDD) algorithms. This interface is based on the Functional Mock-up Interface (FMI), and the Building Automation Java Architecture (Baja) standards. This allows HVAC designers and control vendors a) developing, testing and improving control and FDD algorithms under a wide range of conditions in simulation, b) exporting them in a code which implements the FMI standard, and c) importing them in building management system for deployment or to support building operation.

We anticipate the proposed approach to facilitate the deployment of control and FDD algorithms on building management systems, and the use of simulation models during the operation of the building, such as the real-time comparison of actual performance relative to design intent.

INTRODUCTION

Building owners could save an average of 38% on their heating and cooling bills if they installed energy efficient controls that make their heating, ventilation and air conditioning (HVAC) systems more energy efficient (Wang et al., 2011).

A cost-effective path to improve the energy efficiency of buildings could be achieved by replacing mal-functioning or energy-wasting control algorithms with better ones. Although there is a large number of well-developed control technologies and algorithms available (Hydeman et al., 2003); their adoption is slow. This is correlated with their deployment cost and the risk of mal-functioning due to limited code testing. To deploy such control algorithms, engineering firms and retro-commissioning agents need to be familiar with different building management systems (BMS), each vendor requiring a new implementation. This complicates the implementation and increases cost.

As the demand for low-energy buildings increases, their control and fault detection and diagnostics (FDD) algorithms become more complex. Implementing such algorithms is challenging, since they require mathematical routines and functions that are not provided by standard BMS. There is a strong need for an open standard interface in the software layer for BMS which a) supports the import, and deployment of control and FDD algorithms, and b) facilitates the use of simulation models for building operation.

We selected the Functional Mock-up Interface (FMI) standard as open standard interface. The FMI standardizes an Application Programming Interface (API) that allows control algorithms, FDD algorithms or models of physical processes to be included in simulation programs or control hardware. It enables also the interoperability and exchange of these models across different platforms. We extended a BMS that is based on the Building Automation Java Architecture standard (Baja) with the FMI standard. This interface allows the BMS to import and use any control algorithm embedded in a simulation model which implements this API. Furthermore, it reduces engineering cost associated with the deployment of such algorithms on different BMS as the API is standardized. It provides a robust pathway to use simulation during building operation.

This paper is structured as follows: Section 2 introduces the FMI standard. Section 3 describes the BMS which has been used in this article. Section 4 shows the implementation of the FMI standard in the BMS. Section 5 shows applications which demonstrate the usefulness of this approach. Section 6 summarizes our findings.

FUNCTIONAL MOCK-UP INTERFACE

The FMI standard has originally been developed in the Information Technology for European Advancement (ITEA2) project MODELISAR. The development of the FMI standard has been initiated by Daimler, is being

further developed by 16 companies and research institutes and is currently supported by over 50 programs. See FMI Standard (FMI Standard, 2013) for an exhaustive list of programs.

The FMI standard is intended to support model exchange and co-simulation of dynamic models using a combination of XML[1]-file, optional C-code and/or shared libraries.

The FMI standard version 1.0, which has been used in this contribution, consists of three parts:

a) FMI for model exchange, which standardizes an interface for coupling simulation programs that are integrated in time by an external solver (MODELISAR-Consortium, 2010b).

b) FMI for co-simulation, which standardizes an interface for coupling simulation programs that contain their own solver for time integration (MODELISAR-Consortium, 2010a).

c) FMI for Product Lifecycle Management, which provides a standardized way to handle FMI related data (MODELISAR-Consortium, 2011).

A system model or simulation program which implements the FMI standard is called a Functional Mock-up Unit (FMU). An FMU is implemented as a zip-file. An FMU contains the FMI model description file, which is an XML-file with information needed by an import program, optional C-code and/or shared libraries required to interface with the model or simulation program, and resource files such as data tables (see *Figure 1*).

This contribution uses the FMI for co-simulation application programming interface (API). This API provides two implementations, namely *CoSimulation_Tool* and *CoSimulation_StandAlone*. In the *CoSimulation_Tool* implementation, the FMU contains a wrapper for shared libraries that interact with the slave program so that a master program which imports the FMU can interface with the slave program in a standardized way. In the *CoSimulation_StandAlone* implementation, the FMU contains the model and a solver.

```
modelDescription.xml  // Description of model (required file)
model.png             // Optional image file of model icon
documentation         // Optional directory with documentation
sources               // Optional directory containing all C-sources
binaries              // Optional directory containing the binaries
  win32               // Optional binaries for 32-bit Windows
  <modelIdentifier>.dll   // DLL of the model interface
  win64               // Optional binaries for 64-bit Windows
  ...
  linux32             // Optional binaries for 32-bit Linux
   <modelIdentifier>.so   // DLL of the model interface
   …
  darwin32            // Optional binaries for 32-bit Darwin
   <modelIdentifier>.dylib   // DLL of the model interface
   …
resources             // Optional resources needed by model
```

Figure 1 Structure of an FMU file.

Figure 2 shows how a simulation program, which supports the FMI API, can export a sub-system model as an FMU for model exchange or for co-simulation. In the co-simulation case, the sub-system is exported with a solver for time integration.

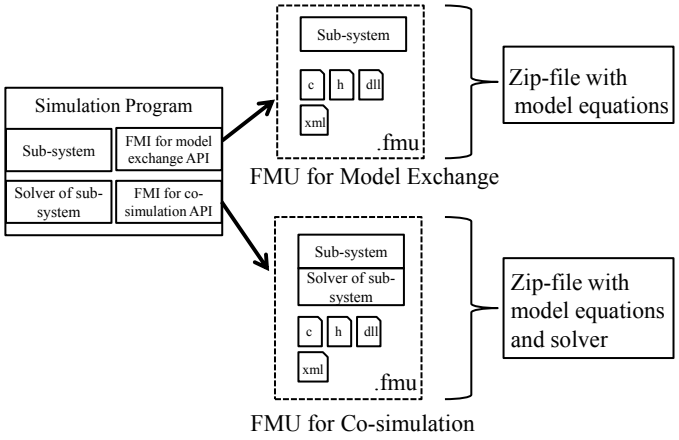

Figure 2 Exporting an FMU for co-simulation or model exchange (MODELISAR-Consortium, 2010a).

State-of-the art of FMI in building community

The FMI standard has been used in the buildings community primarly to link simulation programs for co-simulation. Recent efforts are going on within the IEA EBC Annex 60 (IEA EBC Annex 60, 2012) to develop the next generation computational tools based on the Modelica (Mattsson and Elmqvist, 1997) and the FMI

[1] XML stands for extensive Markup Language.

standards. An FMI for co-simulation has been implemented in WUFI Plus (Pazold et al., 2012) to integrate Modelica models to it. The Institute for the Sustainable Performance of Buildings has been developing a web-based eLearning tool in which a Web interface communicates with FMUs for co-simulation, which are used to simulate heat transfer through building envelope, HVAC systems, control systems and equipment faults of a building, and visualize this response at an interactive web browser through WebGL (Deringer et al., 2012). The University of California at Berkeley and the Lawrence Berkeley National Laboratory (LBNL) have developed an FMU import interface for co-simulation in the Building Controls Virtual Test Bed (Wetter, 2011). LBNL released an FMU import interface for co-simulation in EnergyPlus (Nouidui et al., 2014), and a software utility which exports EnergyPlus as an FMU for co-simulation (Nouidui et al., 2013). The Austria Institute of Technology has developed an FMU import interface in TRNSYS to support the integration of Modelica-based types within TRNSYS (Elsheikh et al., 2013). Fraunhofer IBP is working on developing an FMI in CoSimA+ which is a framework for coupling of distributed, heterogeneous numerical models (Treeck et al., 2011).

Although the FMI standard is increasingly used in the buildings community, there seems to be no related work that integrates this interface in hardware such as BMS. This could provide a pathway for deployment of control and FDD algorithms, and the coupling of simulation models, or programs with measurements for building operation. This contribution aims to address this gap by implementing an FMI in the Niagara^{AX} framework. We selected the Niagara^{AX} framework because of its open-source architecture and its wide use in the buildings industry.

NIAGARA ^{AX} - BUILDING MANAGEMENT SYSTEM

The Niagara^{AX} is an open, Java-based framework which creates a common environment to connect devices of different manufacturers or communication protocols. Niagara^{AX} models the data and behavior of the devices using normalized software components. Niagara^{AX} uses the Common Object Model to normalize any device which is connected to it. The framework takes data elements such as the inputs, outputs, setpoints, schedules, control parameters of devices and converts them into normalized software components. This is achieved by unifying the attributes of the devices, creating a database of objects that can talk to and work coherently with each other. Niagara^{AX} implements the Baja API (Tridium and Sun-Microsystems, 2000) to model the devices. Baja is an open standard developed by Tridium and the Java community. Baja uses the Java API and an XML schema to allow developers to convert multi-vendor device protocols and communication standards with Internet technologies into a single universal standard and adapt them into an open, interoperable environment. Niagara^{AX} supports different protocols such as BACnet (ASHRAE, 1987) and LonWorks (ECHELON, 1999).

FUNCTIONAL MOCK-UP INTERFACE IN NIAGARA^{AX}

Implementation

To import FMUs for co-simulation in Niagara^{AX}, we developed and added a new module called fmu.jar[2] to Niagara^{AX}. This module contains two Java classes called BFMUService and BFMUComponent which allow importing FMUs in the framework, where they can then be linked to other Niagara^{AX} modules.

The two classes BFMUService and BFMUComponent use JFMI (Brooks et al., 2012) to import FMUs in Niagara^{AX}. JFMI is a software package written in Java and co-developed by the authors of this contribution. JFMI allows interfacing Java applications with FMUs for co-simulation or model exchange. For example, JFMI can be used to unzip an FMU, parse the model description file, and perform a co-simulation.

BFMUService

BFMUService inherits BAbstractService, a class which provides different services such as a web service to Niagara^{AX}. These services are automatically started when the framework is launched. In our implementation, BFMUService uses JFMI to import and unzip an FMU, and to retrieve the FMU's properties. BFMUService can be used to load different FMUs in the framework. BFMUService implements two main functions, doImportFMU(), and parseFile() which are described below:

public void doImportFMU(…)

This method imports an FMU which is specified by providing a path to the FMU's location. This method calls functions provided by JFMI to unzip and parse the model description file.

private void parseFile(…)

This method uses JFMI to parse the model description file of the FMU. It then retrieves the names of all FMU

[2] jAR stands for java ARchive which is a package file format used to aggregate many java class files.

variables with the causality "input" and "output". These variables are saved in a hashtable with the key being the name of the FMU and the values being its properties..

BFMUComponent

BFMUComponent inherits BComponent, which is one of the core Baja's types used to normalize objects in Niagara^{AX}. BFMUComponent is used in our implementation to instantiate an FMU in Niagara^{AX}. *Figure 3* shows such an instance. BFMUComponent implements the five major functions doCreatePorts(), doRunSimulation(), changed(), executeFMUComputation(), and doForcedEndSimulation (), which are described below:

public void doCreatePorts(…)

This method retrieves the properties of a given FMU from the hashtable created by BFMUService, and creates a block in the framework which has as many slots[3] as the number of inputs and outputs of the FMU.

public void doRunSimulation(…)

This method is called at the beginning of an FMU simulation. It loads the FMU's shared library and checks whether the FMI functions exist in the library. It then instantiates and initializes the FMU by calling fmiInstantiateSlave() and fmiInitializeSlave() respectively.

public void changed(…)

This method fires whenever an input or a parameter of the BComponent changed. This method determines whether all inputs of the BComponent are provided, in which case it calls executeFMUComputation to trigger the computation.

private void executeFMUComputation(…)

This method takes the inputs of the BFMUComponent, converts them to a format that can be used by the FMU, sets the inputs of the FMU by calling fmiSetReal(), does a time integration by calling fmiDoStep(), retrieves the outputs of the FMU by calling fmiGetReal(), and converts them to a format that can be sent to Niagara^{AX}. At the end of the simulation, this method calls fmiTerminateSlave() to terminate the FMU, and fmiFreeSlaveInstance() to release it.

public void doForcedEndSimulation(…)

This method is used to force the end of the simulation of an FMU. This method checks first if the FMU for which the simulation should be aborted has been previously instantiated. It then calls fmiTerminateSlave() and fmiFreeSlaveInstance() to terminate and release the FMU.

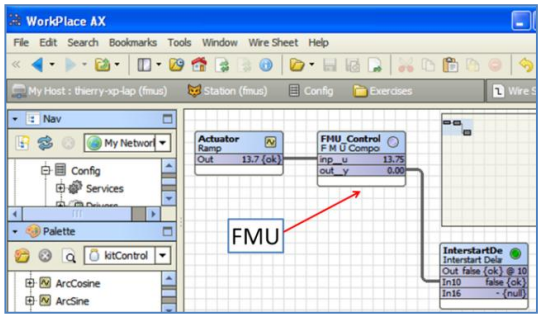

Figure 3 Example of an FMU imported in Niagara^{AX}.

APPLICATIONS

The addition of an FMU import interface in Niagara^{AX} enables various applications. In this section, we present three possible applications.

Monitoring of actual performance relative to design intent

During the design, an HVAC designer creates a simulation model of a building and its HVAC system and controls. He then exports the model as an FMU for co-simulation and imports it to Niagara^{AX}. In Niagara^{AX}, he links the model input to measured data. The design model can then be used to compute expected room air temperature or energy consumption, which in turn can be used to compare measured with expected performance.

Figure 4 shows the model of a building which has been used to demonstrate this use case. The building model is the one used in Case600 of ANSI/ASHRAE Standard 140 (ANSI/ASHRAE, 2007).

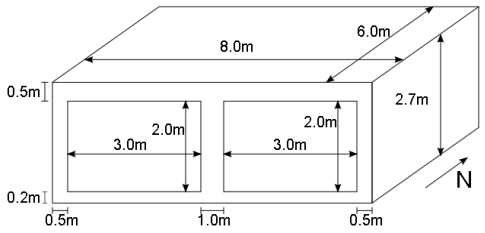

Figure 4 Building model of Case600 (ANSI/ASHRAE, 2007).

As a first step, the HVAC engineer develops a model of the building and its HVAC and control system in Modelica. *Figure 5* shows the Modelica implementation of Case600. The building model has been constructed using component models of the Modelica *Buildings* library (Wetter et al., 2014). The building model consists

[3] A slot is an object port which allows a component in Niagara^{AX} to be connected to other components.

of three main parts, the building envelope (1), an air-based HVAC system (2), and a PI-controller (3) which is used to control the room air temperature.

As a second step, the HVAC engineer simulates the building model along with its HVAC and control system to ensure correctness of the control algorithms.

As a third step, the model of the building with its HVAC and control system is exported as an FMU for co-simulation and imported into NiagaraAX. This model can then run in real-time to predict the expected performance of the building.

Since at the time of this writing, the Niagara^{AX} framework was not physically connected to any device, we used a signal generator in Niagara^{AX} to emulate the room air temperature that could have been measured and made available in Niagara^{AX}. The FMU itself has an output port which reports the simulated room air temperature. These two quantities can be used to compare measured and expected performance. The framework can then use some of its services to report potential discrepancies to the building manager (see *Figure 6*).

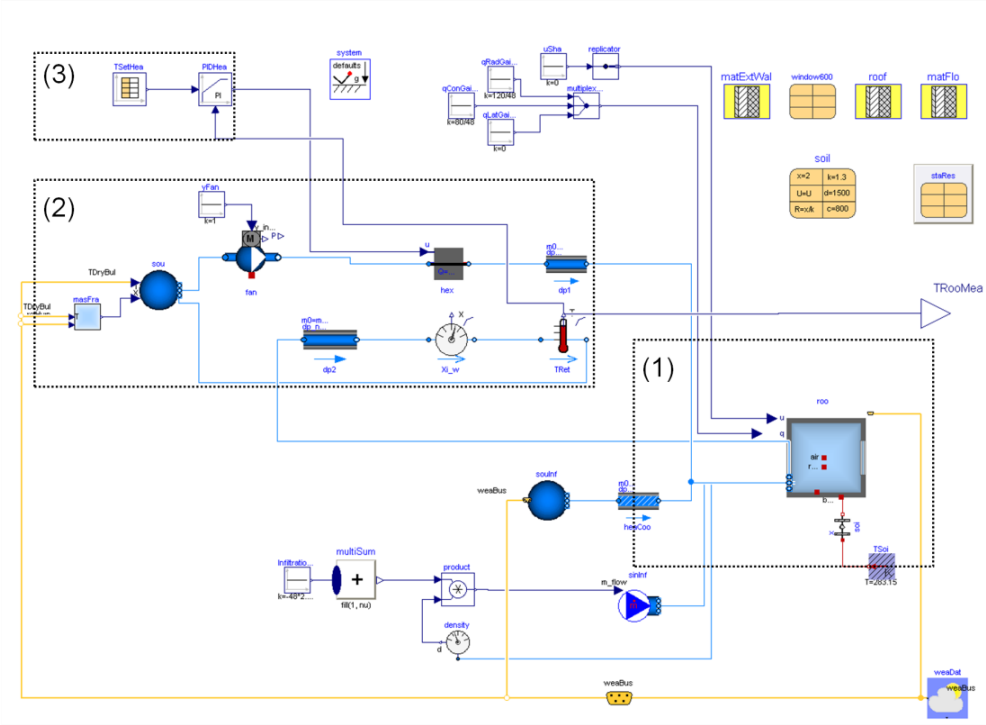

Figure 5 Modelica implementation of building model with HVAC and control system.

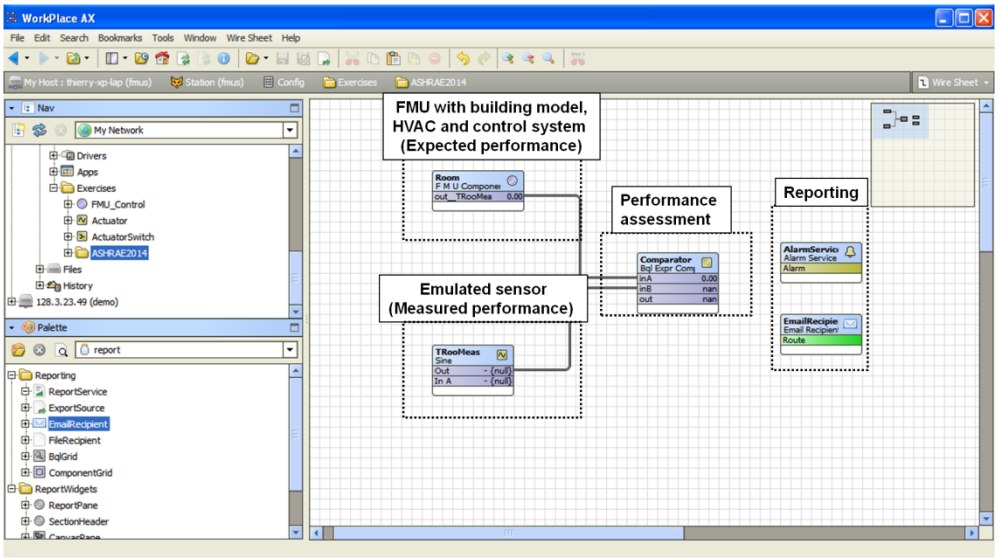

Figure 6 Simulation-based performance monitoring in Niagara^{AX}.

Model-based FDD

A researcher or product developer may develop an FDD algorithm for systems such as Air Handling Units (AHU). In this process, he selects from a library of HVAC components the specific components that will form the different AHUs. Next, he connects them to create the system model, and then enters the information he has collected from the physical system. This information includes the heat flow characteristics of the heating and cooling coils, and the duct pressure drop and flow rate at the design conditions. The developed model will then be calibrated. Once the model is calibrated, it will be integrated into a FDD simulation model, which will support the following analysis:

1. During fault detection, the model will be used to compute the expected sensor values, which are then compared to measured data.
2. During fault diagnostics, the model will emulate the signature of different fault scenarios. The model outputs for these fault scenarios are then compared to the observed measurements in order to diagnose the fault.

The researcher will test this model in simulation, and then export the model as an FMU for co-simulation. This FMU can then be imported in Niagara^{AX} and be linked with an actual building energy system.

Design and deployment of control algorithms

A researcher, product developer or advanced HVAC engineer develops and tests an advanced control algorithm in simulation, exports it as an FMU for co-simulation, and imports it to Niagara^{AX} to link it to an actual building for closed loop control.

CONCLUSION

The Functional Mock-up Interface standard is well suited to deploy control algorithms. We anticipate the integration of FMI in BMS to facilitate the deployment of control algorithms and FDD algorithms that have been tested in simulation, as well as the deployment of simulation models to assist real-time monitoring, FDD and model-based controls. We therefore anticipate the integration of FMI in BMS to facilitate the reuse of simulation models from the design phase to the operation phase of buildings, thereby contributing to closing the performance gap between predicted and actual energy use. Future work will include applications such as hardware-in-the-loop or performance monitoring of buildings where the Niagara^{AX} framework is physically connected to real devices, and linked to simulation tools through its FMI.

ACKNOWLEDGMENTS

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

We thank Pacific Controls for sponsoring the development of a Functional Mock-up Unit for co-simulation import interface in Niagara^{AX}.

REFERENCES

ANSI/ASHRAE 2007. Standard Method of Test for the Evaluation of Building Energy Analysis

Computer Programs (ANSI/ASHRAE Standard 140-2007). Atlanta.

ASHRAE. 1987. *BACnet* [Online]. Available: http://www.bacnet.org/index.html.

Brooks, C., Lee, E. A., Wetter, M., Nouidui, T. S., Broman, D. & Tripakis, S. 2012. *JFMI - A Java Wrapper for the Functional Mock-up Interface* [Online]. Available: http://ptolemy.eecs.berkeley.edu/java/jfmi/.

Deringer, J. J., Nahman, J. E., Heming, K., Wetter, M., Pang, X. F. & Konstantoglou, M. LearnHPB and eLAD – Two Related Online eLearning Platforms for High Performance Buildings. 2012 ACEEE Summer Study on Energy Efficiency in Buildings, 2012 Pacific Grove, CA.

Echelon. 1999. *LonWorks* [Online]. Available: http://www.echelon.com/technology/lonworks/

Elsheikh, A., Widl, E., Palensky, P., Dubisch, F., Brychta, M., Basciotti, D. & Mueller, W. Modelica-enabled rapid prototyping via TRNSYS. 13th International Conference of the International Building Performance Simulation Association, 2013 Chambery, France.

FMI Standard. 2013. *Functional Mock-up Interface Support in Tools* [Online]. Available: https://www.fmi-standard.org/tools [Accessed September 14 2013].

Hydeman, M., Taylor, S., Stein, J., Kolderup, E. & Hong, T. 2003. . Advanced Variable Air Volume: System Design Guide: Design Guidelines. California Energy Commission.

IEA EBC Annex 60. 2012. *New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards* [Online]. Available: http://www.iea-annex60.org/.

Mattsson, S. E. & Elmqvist, H. An international effort to design the next generation modeling language. 7th IFAC Symposium on Computer Aided Control Systems Design, 1997 Gent, Belgium.

Modelisar-Consortium. 2010a. *Functional Mock-up Interface for Co-Simulation* [Online]. Available: https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_CoSimulation_v1.0.pdf 2013].

Modelisar-Consortium. 2010b. *Functional Mock-up Interface for Model-Exchange* [Online]. Available: https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_ModelExchange_v1.0.pdf 2013].

Modelisar-Consortium. 2011. *Functional Mock-up Interface for Product Lifecycle Management* [Online]. Available: https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_PLM_v1.0.pdf 2013].

Nouidui, T. S., Lorenzetti, D. M. & Wetter, M. 2013. *EnergyPlusToFMU* [Online]. Available: http://simulationresearch.lbl.gov/fmu/EnergyPlus/export/index.html.

Nouidui, T. S., Wetter, M. & Zuo, W. 2014. Functional Mock-up Unit for Co-Simulation Import in EnergyPlus. *Journal of Building Performance Simulation,* 7**,** 192-202.

Pazold, M., Burhenne, S., Radon, J., Herkel, S. & Antretter, F. 2012. Integration of Modelica models into an existing simulation software using FMI for Co-Simulation. 9th International Modelica Conference, September 03 2012 Munich, Germany.

Treeck, C. V., Stratbuecker, S., Bolineni, S. R., Schmidt, C. & Woelki, D. Coupling heteregeneous computational codes for human-centred indoor thermal performance analysis. 11th International Conference of the International Building Performance Simulation Association, 2011 Sydney, Australia.

Tridium & Sun-Microsystems. 2000. *Baja: A Java - based Architecture Standard for the Building Automation Industry* [Online]. Available: http://www.automatedbuildings.com/wsim/Baja_White_Paper.pdf 2013].

Wang, W., Huang, Y., Katipamula, S. & Brambley, M. R. 2011. Energy Savings and Economics of Advanced Control Strategies for Packaged Air-Conditioning Units with Gas Heat.

Wetter, M. 2011. Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed. *Journal of Building Performance Simulation,* 4**,** 185-203.

Wetter, M., Zuo, W., Nouidui, T. S. & Pang, X. 2014. Modelica Buildings library. *Journal of Building Performance Simulation,* 7**,** 253-270.